Session 9: Dependency Management with Maven

Fortgeschrittene Programmierung (Java 2)

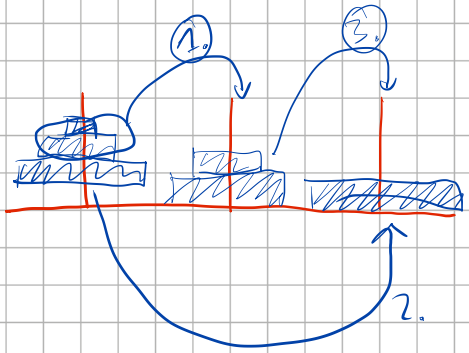
Nils Reiter
nils.reiter@uni-koeln.de

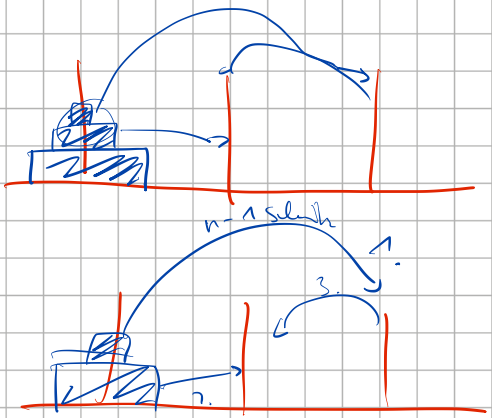
June 14, 2022

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Recap: Recursion

- Two components
  - Recursion step: How to make one additional step
  - Base case(s): When and how to stop doing additional steps
- Two relevant areas in programming
  - Recursive data structures – how we store things
  - Recursive algorithms – how we process things
- Trees: Recursive data structures commonly used for all kinds of hierarchies





movePieces(3, l, r, m)

↳ movePieces(2, l, m, r)

↳ movePieces(1, l, r, m)

Section 1

Apache Maven

Introduction

- Maven: A build tool
  - Build process: Convert source code to binary code
  - More than pure compilation: Libraries (in correct versions), ressources, packages, documentation, testing, ...
  - Make build environment explicit
    - And ensure it's the same for every developer
- Build process often automated (continuous integration, CI)
  - Maven can also run on GitHub to perform automatic testing
- Documentation: `https://maven.apache.org/guides/`

How to use Maven

Two core ingredients

File `pom.xml`

- "Project Object Model"
- XML-file
- Contains all meta data about the build process
- `groupId`/`artifactId`/`version` identifies a Maven artifact

How to use Maven

Two core ingredients

File `pom.xml`

- "Project Object Model"
- XML-file
- Contains all meta data about the build process
- `groupId`/`artifactId`/`version` identifies a Maven artifact

Standard Directory layout

- pom.xml
- src
  - main
    - java
    - resources
  - test
    - java
    - resources
- target

Repositories

Local

- Maven maintains a local repository in your home directory
  - Unix: `~/.m2`
  - Windows: `C:\Users\<USERNAME>\.m2`
- Dependencies are installed in this directory
- Classpath is maintained to include the necessary directories
- Local repository can become quite large (my laptop: 7GB), cleanup from time to time

Repositories

Local

- Maven maintains a local repository in your home directory
  - Unix: `~/.m2`
  - Windows: `C:\Users\<USERNAME>\.m2`
- Dependencies are installed in this directory
- Classpath is maintained to include the necessary directories
- Local repository can become quite large (my laptop: 7GB), cleanup from time to time

Remote

- Central repository: Filled by the maven community (maintained by Sonatype)
  - `https://search.maven.org`
- Organisations may maintain their own internal repositories

Build Lifecycle

Maven Documentation: The Build Lifecycle

Lifecycle	Description
`validate`	All necessary information present
`compile`	Compile source code
`test`	Run unit tests on the compiled code
`package`	Package into a jar file
`verify`	Run tests on the jar file
`install`	Install into the local maven repository
`deploy`	Copy package to a remote maven repository

Table: Most important Maven default lifecycle phases

Build Lifecycle

Maven Documentation: The Build Lifecycle

Lifecycle	Description
`validate`	All necessary information present
`compile`	Compile source code
`test`	Run unit tests on the compiled code
`package`	Package into a jar file
`verify`	Run tests on the jar file
`install`	Install into the local maven repository
`deploy`	Copy package to a remote maven repository

Table: Most important Maven default lifecycle phases

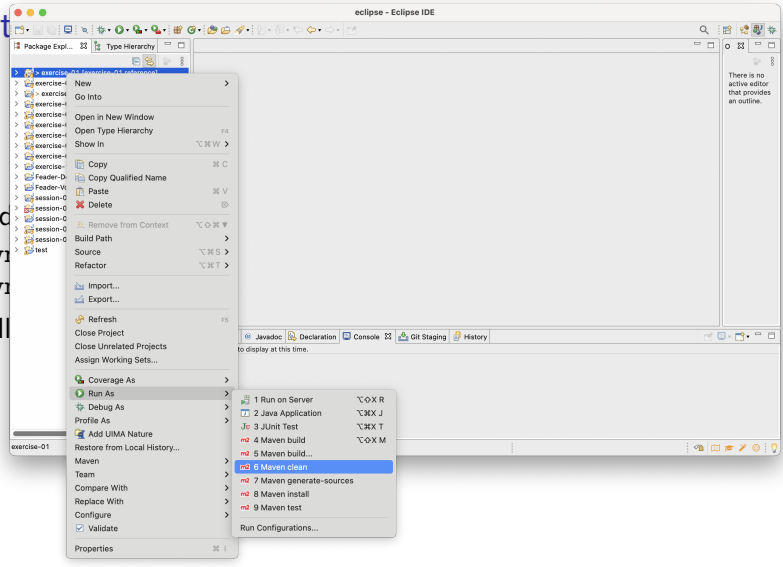
Other lifecycles: `clean` and `site`

Building with Maven

- Command line:
  - `$ mvn package` to create a jar file
  - `$ mvn clean compile` to remove old compilations and compile again
- Technically, each phase is handled by a different plugin

Building wit

- Command
  - `$ mvn`
  - `$ mvn`
- Technicall



pom.xml Sections

Properties

```
<project ...>
  ...
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>
  ...
</project>
```

- Properties are specified in top-level `properties`-element
- Some properties are project-global (`project.build.sourceEncoding`)
- Some properties specific to some plugins (`maven.compiler.target`)

Maven Documentation: POM

pom.xml Sections

Build

```
<project ...>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>GROUPID OF PLUGIN</groupId>
        <artifactId>ARTIFACT ID OF PLUGIN</artifactId>
        <version>VERSION OF PLUGIN</version>
        <configuration>PLUGIN CONFIGURATION PARAMETERS</configuration>
        <executions>
          <execution>
            <id>SOME ID</id>
            <goals>
              <goal>SOME FUNCTION OF THE PLUGIN</goal>
            </goals>
          </execution>
        </executions></plugin><plugin>...</plugin></plugins></build>
  ...
</project>
```

pom.xml Sections

Build

- Allows to change each lifecycle
- Example use cases from annotation tool CorefAnnotator
  - Generate source code from XML files (before compiling)
  - Generate test resources
  - Exclude specific resources from packaging
  - Compile other languages
  - `https://github.com/nilsreiter/CorefAnnotator/blob/master/pom.xml`

pom.xml Sections

Dependencies

```
1  <project ...>
2    ...
3    <dependencies>
4      <dependency>
5        <groupId>org.glassfish.jaxb</groupId>
6        <artifactId>jaxb-runtime</artifactId>
7        <version>2.3.2</version>
8      </dependency>
9      <dependency>...</dependency>
10   </dependencies>
11   ...
12 </project>
```

- Top-level element
- Specify each dependency as triple `groupId`, `artifactId`, and `version`

pom.xml Sections

Dependencies

```
1  <project ...>
2    ...
3    <dependencies>
4      <dependency>
5        <groupId>org.glassfish.jaxb</groupId>
6        <artifactId>jaxb-runtime</artifactId>
7        <version>2.3.2</version>
8      </dependency>
9      <dependency>...</dependency>
10   </dependencies>
11   ...
12 </project>
```

- Top-level element
- Specify each dependency as triple `groupId`, `artifactId`, and `version`
- All dependencies in maven universe!

Additional elements

- `scope`: compile/provided/runtime/test/system
- `classifier`: Arbitrary string to distinguish variants
- `type`: jar/war/ejb-client/...

pom.xml Sections

Dependencies

Specification	Description
`1.0`	1.0 if no other version appears earlier in dependency tree
`[1.0]`	1.0 and only 1.0.
`(,1.0]`	Any version $<=$ 1.0
`[1.2,1.3]`	Any version between 1.2 and 1.3, inclusive
`[1.0,2.0)`	Any version between 1.0 inclusive and 2.0 exclusive
`[1.5,)`	Any version greater than or equal to 1.5
`(,1.0],[1.2,)`	Any version $<=$ 1.0 or $>=$ 1.2, but not 1.1
`(,1.1),(1.1,)`	Any version except 1.1

Table: Specifying versions for dependencies in Maven. Works best with semantic versioning

Semantic Versioning

$$MAJOR.MINOR.PATCH$$

Given a version number MAJOR.MINOR.PATCH, increment the:

- MAJOR version when you make incompatible API changes,
- MINOR version when you add functionality in a backwards compatible manner, and
- PATCH version when you make backwards compatible bug fixes.

Additional labels for pre-release and build metadata are attached

- E.g. 2.0.0-beta

Semantic Versioning

pom.xml

Properties

```
1  <project>
2    <properties>
3      <version.javafx>14.0.2.1</version.javafx>
4    </properties>
5    ...
6    <dependencies><dependency>
7        <groupId>org.openjfx</groupId>
8          <artifactId>javafx-controls</artifactId>
9          <version>${version.javafx}</version>
10       </dependency><dependency>
11         <groupId>org.openjfx</groupId>
12         <artifactId>javafx-swing</artifactId>
13         <version>${version.javafx}</version>
14       </dependency></dependencies>
15 </project>
```

- Properties can be used throughout the POM
- Can be defined in other ways as well: Profiles, inheritance, default, system OS, ...

Properties

Finding Libraries

- The usual places: Recommendations, mailing lists, research papers, stackoverflow, search engines
- ⚠ Switching from one library to another might be costly
- Make conscious decisions about libraries
  - Is it open source?
  - How large is the community?
  - Is it in active development?
  - How well is it documented?

Finding Libraries

- The usual places: Recommendations, mailing lists, research papers, stackoverflow, search engines
- ⚠ Switching from one library to another might be costly
- Make conscious decisions about libraries
  - Is it open source?
  - How large is the community?
  - Is it in active development?
  - How well is it documented?
- Finding a maven artifact: `https://search.maven.org`

Finding Libraries

- The usual places: Recommendations, mailing lists, research papers, stackoverflow, search engines
- ⚠ Switching from one library to another might be costly
- Make conscious decisions about libraries
  - Is it open source?
  - How large is the community?
  - Is it in active development?
  - How well is it documented?
- Finding a maven artifact: `https://search.maven.org`
- ⚠ Beware: It's foreign code, there is no automatic security checking
  - "Software Supply Chain Attack"

Inheritance

- Each POM inherits from its parent POM
- Root-POM:
  `https://maven.apache.org/ref/3.6.3/maven-model-builder/super-pom.html`

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.mycompany.app</groupId>
    <artifactId>my-app</artifactId>
    <version>1</version>
  </parent>
  <artifactId>my-module</artifactId>
</project>
```

- This POM inherits `groupId` and `version` from its parent

Inheritance

What is inherited by a child POM?

- dependencies
- developers and contributors
- plugin lists (including reports)
- plugin executions with matching ids
- plugin configuration
- resources

demo

Use Maven to add Apache Commons IO and make `Document` easier

Maven and Git

- Maven controls all settings needed for building
- Eclipse can derive its settings from Maven
  - ...and other IDEs (Netbeans etc.) too!
- ➔ thus, Eclipse settings do not belong in the git repository
- Team members can work with different IDEs

Maven and Git

- Maven controls all settings needed for building
- Eclipse can derive its settings from Maven
  - ...and other IDEs (Netbeans etc.) too!
- ➔ thus, Eclipse settings do not belong in the git repository
- Team members can work with different IDEs
- Future exercises will come in the form of maven projects

Summary

- Maven to organise your dependency
- Works across computers
  - because dependencies are downloaded automatically from Maven central
- Formal definition of the build process
  - Replication
  - Documentation

Exercise

https://github.com/idh-cologne-java-2-summer-2023/exercise-10